# Emu: Engagement Modeling for User Studies

**Bo-Jhang Ho**
University of California, Los Angeles
Los Angeles, CA 90095, USA
bojhang@ucla.edu

**Nima Nikzad**
Scripps Translational Science Institute
La Jolla, CA 92037, USA
nnikzad@scripps.edu

**Bharathan Balaji**
University of California, Los Angeles
Los Angeles, CA 90095, USA
bbalaji@ucla.edu

**Mani Srivastava**
University of California, Los Angeles
Los Angeles, CA 90095, USA
mbs@ucla.edu



*UbiComp/ISWC'17 Adjunct*, September 11-15, 2017, Maui, HI, USA © 2017 Copyright is held by the owner/author(s).
ACM ISBN 978-1-4503-5190-4/17/09.
https://doi.org/10.1145/3123024.3124568

## Abstract
Mobile technologies that drive just-in-time ecological momentary assessments and interventions provide an unprecedented view into user behaviors and opportunities to manage chronic conditions. The success of these methods rely on engaging the user at the appropriate moment, so as to maximize questionnaire and task completion rates. However, mobile operating systems provide little support to precisely specify the contextual conditions in which to notify and engage the user, and study designers often lack the expertise to build context-aware software themselves. To address this problem, we have developed *Emu*, a framework that eases the development of context-aware study applications by providing a concise and powerful interface for specifying temporal- and contextual-constraints for task notifications. In this paper we present the design of the *Emu* API and demonstrate its use in capturing a range of scenarios common to smartphone-based study applications.

## Author Keywords
Context-aware; mobile applications; engagement; just-in-time assessment; push notifications; user studies; mhealth

## ACM Classification Keywords
H.5.m [Information interfaces and presentation (e.g., HCI)]: Miscellaneous; D.3.3 [Language Constructs and Features]: Frameworks

## Introduction

In recent years, many behavior and health related studies have leveraged mobile apps to improve study protocol adherence and participant engagement [11, 16]. Notifications from such apps can help with adherence to study requirements - such as updating a food diary after each meal or measuring blood pressure after exercise - while reducing the cognitive load associated with participation in such studies [11]. However, phone users already receive a mean of 63 notifications per day [15], and ill-timed interruptions can be distracting and affect productivity [2]. To maximize participation value and improve study adherence, notifications must be both contextually relevant and timely [5].

While prior works have presented robust and efficient context-recognition algorithms [8], adoption of these techniques to drive study-related notifications has been slow. Specifying contexts precisely is challenging because: (i) keeping track of interrelated event- and context-driven tasks can be complex and error prone, and (ii) there is no interface for tracking user behavior and adjust app logic. Thus in practice, most such applications are simply driven by time-based constraints, e.g. notify to log food at 8AM and 6PM [5, 11].

To address these challenges, we introduce *Emu*, a framework for developers to concisely and precisely specify when and how to engage users. Emu relieves the developer of the burden of tracking contextual states and delivers notifications when the required conditions are met. Emu tracks responses to notifications and schedules future notifications accordingly. We show that Emu can concisely capture a wide variety of scenarios common to user studies.

## Building a Context-aware Study App

We consider three stakeholders in the life cycle of a study app: (i) a **study designer**, the domain expert conducting the study, (ii) a **developer**, who translates study specifications into code, and (iii) a **user**, the study participant. In our example study, the designer would like to collect blood pressure (BP) measurements after each workout to monitor the health status of hypertensive patients. These requirements translate to detailed specifications for the developer.

For simplicity, we only consider running as workout. When a run session exceeds 15 minutes, the study app will notify the user to take a BP measurement. Each user is provided a Bluetooth enabled BP monitor that communicates with the study app, and the app will only remind the user to take measurements if they are close to the BP instrument within an hour of their run session. If the user ignores the notification, the app will retry displaying the notification two times. If the user fails to measure BP for a week, an encouragement is sent to comply with the study requirements. This example highlights just some of the features that a robust study app must support, and is informed by literature review and informal discussions with study designers.

*Challenges in Scheduling Notifications*

Smartphone OSes and third-party libraries provide some support for building such apps. Android and iOS support detection of contexts such as running and proximity to Bluetooth devices, and Apple ResearchKit[1] assists development of questionnaires and intervention tasks. But, scheduling of notifications is limited to time or location based reminders.

**Matching complex events:** Consider the condition for sending a notification as running for 15 minutes. The app needs to start a countdown timer when the user starts running. If the user stops running before the timer hits zero, the timer is invalidated; otherwise, the notification is sent. The complexity increases as we add more conditions: if the

[1] http://researchkit.org/

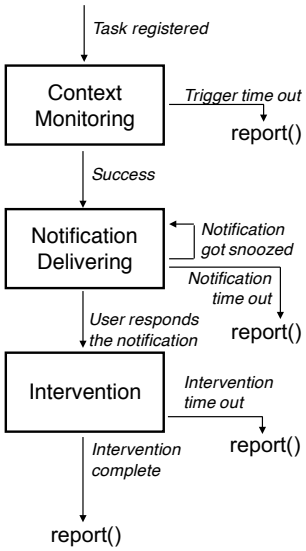


**Figure 1:** Life of a task.

study accepts either running for 15 minutes or walking for 1 hour, then two independent timers will need to monitor each activity. When the condition requires ordered events, such as a run session followed by closeness to BP machine within 1 hour, the app needs to implement the transition logic and maintain timers accordingly. Maintaining different timers is akin to parallel programming and is known to be difficult and error prone. Emu handles all of the timing and condition matching for the developer.

**Monitoring the user:** If the user doesn't goes for a run or gets close to the BP machine, then the app needs to check this condition to nudge the user to comply. If the user ignores the notification, the app needs to monitor it's status and send a reminder. When the user successfully measures the BP after a notification is sent, the app should stop creating more notifications. Emu keeps track of all these aspects for the developer and reports the status with a single callback after the task is finished.

**Historical matching:** Sometimes the condition may depend on previous activities. In our example, a notification is sent if the user does not take BP measurements for a week. The developer needs to create a database to store this information and update it each time user measures BP. Emu automatically logs these events and queries the database internally for developers.

## Related Work

Prior works have studied interruptibility extensively [6, 9, 14]. These works focus on identifying interruptibility from sensor data [6], prior interactions [9], physical activities [13] and other contexts [14]. However, these works do not consider application specific requirements for engaging the user. Bainomugisha et al. [3] propose a language for programming context-aware apps. However, their system does not support timing constructs essential for scheduling notifications. STFL [4] proposes a spatio-temporal framework to specify contextual triggers and is closest to our work. STFL is a Python library and is an independent system. In contrast, Emu is designed for interruptibility using smartphones and wearables. Hence, Emu keeps track of user actions, snoozes notifications, logs events for querying.

## Emu: Framework Overview

Design of Emu was informed by a thorough literature survey, prototype building, and interviews with two study designers experienced in developing seven research studies that utilized mobile apps for monitoring of subjects. This section presents the design and features of Emu.

*Emu Tasks*

To 'register' a notification, a developer specifies: (i) when the notification should be triggered, e.g., at 6PM, or when the user's heart rate is high, (ii) the content of the notification, e.g. a message of encouragement, and (iii) what action to take when the notification is selected or replied to. While specifying the content of a notification is straightforward via the standard APIs, managing the timing of notifications is not: such code is typically event-driven, requiring callback functions that track changes in user context and time, and respond accordingly. Emu bundles the aforementioned components and calls it a **`Task`**. The life cycle of a `Task` is summarized in Figure 1.

The `TaskBuilder` class is used to specify the details relevant to a task's content, contextual trigger, and any actions to trigger when replied to. To tackle the complexity of specifying variety of options in a task, we use the the *builder* design pattern[2]. With this pattern, the developer can concisely create the task in a flexible, step by step manner.

[2] https://en.wikipedia.org/wiki/Builder_pattern

**.repeat(frequency):** Specify the periodicity of the Task. If omitted, it is a one-time Task.

**.interval(period):** The effective period of the Task.

**.when(condition):** A context required to display the Task.

**.then(condition):** An (optional) context to wait for, after **when** has been satisfied, before triggering the Task.

**.notify(text, param):** The content of the notification. Parameters include the priority, number of times allowed to snooze the notifications, and the retry interval.

**.launch(view):** The view presents the requirements, for instance, interventions. The view appears when the user clicks the notification.

**.report(callback):** Specify the callback function to report task results to.

**.startTask(taskId):** Register the Task to the Task Manager.

**Figure 3:** Summary of the fields in Task class.

```
Task taskBP = TaskBuilder.create()
  .repeat('every day')
  .when('walking for 1 hours
        or running for 15 minutes'
  .then('nearBPMachine within 1 hours')
  .notify('Measure blood pressure',
          PRIORITY_MEDIUM,
          'snooze 2 times', 'every 15 mins')
  .launch(bpActivity)
  .report(bpCallBack, 'timeout 2 hours')
  .startTask(BP_TASK_ID);
```

**Figure 2:** Example code of registering a task in Emu. The bold font are reserved keywords in Emu.

Developers can register a repeating task by specifying the frequency (e.g., "every 2 days") via the **repeat()** method. The **interval()** method gives the time period in which notification can be sent, e.g., "10am to 12pm". The **when()** and **then()** methods specify the contextual condition, such as "walking for 1 hour or running for 10 minutes". The **notify()** method specifies the content of a notification and options such as priority and the number of times to repeat the notification if it is not acted on. After the user responds to the notification, the view (or Activity in Android) given in **launch()** will appear. **report()** specifies the callback method to which the result of the Task (completed, ignored, failed to trigger, etc.) is provided. Finally, **startTask()** registers the task with an ID, with which the developer can modify or check the status of a Task later. Figure 3 summarizes the purposes of Emu methods, and Figure 2 implements the blood pressure example using TaskBuilder.

We designed the Task structure to be flexible for developers. They can skip **when()** if they only want randomized time-based notifications. They can skip **notify()** for internal context tracking if user action is not required. **report()** can be used to activate other tasks based on user actions.

*Contextual Triggers*

Emu uses **when()** and **then()** to support complex contextual conditions. The condition can be a boolean expression with **and**, **or**, and **not** operators, e.g. "`running` **and** `heartRate > 100`". Developer can specify the duration for which a condition must hold using the **for** keyword, e.g. "`walking` **for** `10 minutes`". The developer can choose to log events such as BP measurements and query historical values, e.g. "`BPMeasurements < 2` **in the past** `1 week`". A sequence of contextual triggers, each evaluated only after the previous is satisfied, can be specified with **when()** followed by **then()**. There can be multiple **then()** clauses in a task for a sequence of conditions. Contexts included in the boolean expression are monitored in parallel and the Task is triggered when the expression evaluates to true.

*System Architecture*

Figure 4 shows the system architecture of Emu. When a task is started, the Query Parser extracts the string parameters and uses timed automata [1] to unambiguously encode the contextual conditions and transitions to generate a Task object, and registers it with the Task Manager. To evaluate the status of the task, Task Manager maintains several clocks to trace the states of the automata. The Task Manager makes use of the User Preference Manager to account for user preferences (e.g. "Don't make sounds while at work") and Presentation Manager to decide the style of notification (e.g. auditory channel via Alexa). All sensing and context inference is handled by various Sensing Modules, which are simply wrappers around existing OS- and library-provided context inference modules, derived from both built-in phone sensors (e.g. walking derived from inertial sensors) or external IoT devices (e.g. BP machine). Emu records all the task reports and developer specified events along with their timestamps in the Event Database. Historical queries are directed to this database.

| | Description |
|---|---|
| Naughton et al [12] | Provide just-in-time intervention when a smoker enters a smoking area. |
| A-CHESS [7] | Help people quit alcohol. Relaxation instruction are provided when near liquor stores or bars. |
| MH$^2$ [10] | Aim for delivering therapy to ADHD kids. Time-based reminder for surveys and taking medicine. |
| SitCoach [17] | When prolonged sitting is detected, users are asked to walk for 10 minutes. |

**Table 1:** Examples of app-based studies whose behavior can be concisely expressed using the Emu framework.

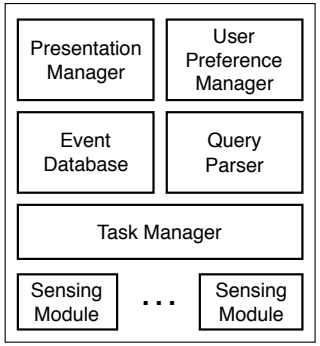


**Figure 4:** System architecture.

## Discussion and Future Work

In our literature review, we identified a wide range of studies that leveraged mobile apps to assess subject behavior, provide health-related interventions, and keep subjects engaged. We provide details on 22 such studies in a separate document due to lack of space[3]. A subset of such studies are listed in Table 1. Emu can successfully capture the scenarios presented by these studies and reduce code complexity for the developer.

Returning to our earlier example of requiring a user to take a blood pressure measurement after running, an implementation of the logic that manages notifying users to take measures at the appropriate time consisted of 20 lines with the Emu framework. A native Android implementation required more code (62 lines) and much higher complexity (due to managing timers).

In this paper we introduced the interface and architecture of the Emu framework, which allows app developers to concisely and precisely specify when and how to engage their users. In future work, we will implement Emu and evaluate its ease of use and performance characteristics.

## Acknowledgements

We thank Monowar Hossain, Alethea Marti, Santosh Kumar and our reviewers for their feedback. This research is funded in part by the National Science Foundation under awards # IIS-1636916 and ACI-1640813, and by the NIH Center of Excellence for Mobile Sensor Data to Knowledge under award # 1U54EB020404-01. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the funding agencies.

[3]User study papers from literature review: https://goo.gl/QujviY

## REFERENCES

1. Rajeev Alur and David L Dill. 1994. A theory of timed automata. *Theoretical computer science* 126, 2 (1994), 183–235.
2. Brian P Bailey, Joseph A Konstan, and John V Carlis. 2001. The Effects of Interruptions on Task Performance, Annoyance, and Anxiety in the User Interface.. In *Interact*, Vol. 1. 593–601.
3. Engineer Bainomugisha, Jorge Vallejos, Coen De Roover, Andoni Lombide Carreton, and Wolfgang De Meuter. 2012. Interruptible context-dependent executions: a fresh look at programming context-aware applications. In *Proc. of the ACM international symposium on New ideas, new paradigms, and reflections on programming and software*. ACM, 67–84.
4. Athanasios Bamis and Andreas Savvides. 2009. STFL: a spatio temporal filtering language with applications in assisted living. In *Proc. of the 2nd International Conference on PErvasive Technologies Related to Assistive Environments*. ACM, 5.

5. Mary Czerwinski, Ran Gilad-Bachrach, Shamsi Iqbal, and Gloria Mark. 2016. Challenges for designing notifications for affective computing systems. In *Proc. of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*. ACM, 1554–1559.
6. James Fogarty, Scott E Hudson, Christopher G Atkeson, Daniel Avrahami, Jodi Forlizzi, Sara Kiesler, Johnny C Lee, and Jie Yang. 2005. Predicting human interruptibility with sensors. *ACM Transactions on Computer-Human Interaction (TOCHI)* 12, 1 (2005), 119–146.
7. David H Gustafson, Fiona M McTavish, Ming-Yuan Chih, Amy K Atwood, Roberta A Johnson, Michael G Boyle, Michael S Levy, Hilary Driscoll, Steven M Chisholm, Lisa Dillenburg, and others. 2014. A smartphone application to support recovery from alcoholism: a randomized clinical trial. *JAMA psychiatry* 71, 5 (2014), 566–572.
8. Hong Lu, Jun Yang, Zhigang Liu, Nicholas D Lane, Tanzeem Choudhury, and Andrew T Campbell. 2010. The Jigsaw continuous sensing engine for mobile phone applications. In *Proc. of the 8th ACM conference on embedded networked sensor systems*. ACM, 71–84.
9. Abhinav Mehrotra, Robert Hendley, and Mirco Musolesi. 2016. PrefMiner: mining user's preferences for intelligent mobile notification management. In *Proc. of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 1223–1234.
10. Lisa M. Mikesell, Alethea F. Marti, Jennifer R. GuzmÃąn, Michael McCreary, and Bonnie Zima. In review. Communicative Uses of mHealth Technology during Early ADHD Stimulant Medication Titration In-Office Visits. *Journal of Applied Communication Research* (In review).
11. Inbal Nahum-Shani, Shawna N Smith, Bonnie J Spring, Linda M Collins, Katie Witkiewitz, Ambuj Tewari, and Susan A Murphy. 2016. Just-in-Time Adaptive Interventions (JITAIs) in mobile health: key components and design principles for ongoing health behavior support. *Annals of Behavioral Medicine* (2016), 1–17.
12. Felix Naughton, Sarah Hopewell, Neal Lathia, Rik Schalbroeck, Chloë Brown, Cecilia Mascolo, Andy McEwen, and Stephen Sutton. 2016. The feasibility of a context sensing smoking cessation smartphone application (Q Sense): a mixed methods study. (2016).
13. Tadashi Okoshi, Julian Ramos, Hiroki Nozaki, Jin Nakazawa, Anind K Dey, and Hideyuki Tokuda. 2015. Reducing users' perceived mental effort due to interruptive notifications in multi-device mobile environments. In *Proc. of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 475–486.
14. Veljko Pejovic and Mirco Musolesi. 2014. InterruptMe: designing intelligent prompting mechanisms for pervasive applications. In *Proc. of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 897–908.
15. Martin Pielot, Karen Church, and Rodrigo De Oliveira. 2014. An in-situ study of mobile phone notifications. In *Proc. of the 16th international conference on Human-computer interaction with mobile devices & services*. ACM, 233–242.
16. Saul Shiffman, Arthur A Stone, and Michael R Hufford. 2008. Ecological momentary assessment. *Annual Review of Clinical Psychology* 4 (2008), 1–32.
17. Saskia Van Dantzig, Gijs Geleijnse, and Aart Tijmen van Halteren. 2013. Toward a persuasive mobile application to reduce sedentary behavior. *Personal and ubiquitous computing* 17, 6 (2013), 1237–1246.